ECMAScript 有两种开发模式：

1. 函数式 ( 过程化 )，2. 面向对象 (OOP)。

面向对象的语言有一个标志，那就是类的概念，而通过类可以创建任意多个具有相同属性和方法的对象。但是，ECMAScript 没有类的概念，因此它的对象也与基于类的语言中的对象有所不同。

面向对象三个基本特征：封装、继承、多态。

对象由属性和方法组成，属性对应变量，表示对象的基本特征，是静态的，方法对应函数，表示对象的行为，是动态的。

**实例创建对象**

实例创建对象，使用 new Object 获取到一个实例对象，然后再给这个对象添加属性和方法。

例：创建一个对象，然后给这个对象新建属性和方法。

var box = new Object();

box.name = 'ujiuye';

box.age = 18;

box.run = function() {

return this.name + this.age;

};

console.log(box.run());

创建了一个对象，并且添加对象的属性和方法，在 run() 方法里的 this，就是代表box 对象本身。这种是 JavaScript 创建对象最基本的方法，但有个缺点，想创建一个类似的对象，就会产生大量的代码。

var box2 = new Object();

box2.name = '小明';

box2.age = 200;

box2.run = function() {

return this.name + this.age;

};

alert(box2.run());

为了解决多个类似对象声明的问题，我们可以使用一种叫做工厂模式的方法，这种方法就是为了解决实例化对象产生大量重复的问题。

**字面量方式创建对象**

把所有的方法和属性都放在 {} 里面，这种方式又叫做字面量创建对象

var obj = {

name: '如花',

age: 18,

run: function() {

alert(this.name);

}

}

console.log(obj.name, obj.age);

obj.run(); // 如花

**工厂模式创建对象**

工厂模式创建对象，是使用函数将创建对象的代码包裹起来，添加好属性和方法后，将这个对象返回出去。

function createObject(name, age) {

var obj = new Object();

obj.name = name;

obj.age = age;

obj.run = function() {

return this.name + this.age;

};

return obj;

}

var box1 = createObject('Lee', 100); // 第一个实例

var box2 = createObject('Jack', 200); // 第二个实例

alert(box1.run());

alert(box2.run());

工厂模式的问题：

工厂模式解决了重复实例化的问题，但还有一个问题，那就是识别问题，因为根本无法搞清楚他们到底是哪个对象的实例。

alert(typeof box1); //Object 判断对象类型

alert(box1 instanceof Object); //true 判断一个对象是否是某个类型

**构造函数创建对象**

构造函数创建对象，可以明确区分对象的种类。

function Person(name, age, job) {

this.name = name;

this.age = age;

this.job = job;

this.sayName = function() {

alert(this.name);

};

}

// 实例对象

var person1 = new Person("小明", 29, "学生");

var person2 = new Person("小张", 27, "程序员");

要创建 Person的新实例，必须使用 new 操作符。以这种方式调用构造函数实际上会经历以下 4个步骤：

(1) 创建一个新对象；

(2) 将构造函数的作用域赋给新对象（因此 this 就指向了这个新对象）；

(3) 执行构造函数中的代码（为这个新对象添加属性）；

(4) 返回新对象。

构造函数的问题：

alert(person1.sayName == person2.sayName); //false 判断的内存地址

同一个方法，因为对象不同，所存储的位置也不同，如果有多个对象，则要分配存储多次，占内存。

**原型创建对象**

原型：每个对象都会有自己的原型，是最顶层类型定义的方法和属性。如 Number 定义了很多的方法和属性，则所有的 number 对象都能使用这个方法和属性。

var num = new Number();

\_\_proto\_\_ 是实例对象的一个属性，指向原型。原型需要用类型名调用，如 Number.prototype。

console.log(num.\_\_proto\_\_ == Number.prototype); // true

function Person() {} // 定义一个构造函数

// 使用原型添加属性和方法

Person.prototype.name = '小明';

Person.prototype.run = function() {

alert(this.name);

};

//prototype: 通过原型添加的方法和属性都是共享的，每一个实例对象都能够使用

var s1 = new Person();

var s2 = new Person();

console.log(s1.name, s2.name); // 小明 小明

console.log(s1.run === s2.run); //true 原型方法同一个

问题：原型创建的对象的属性不能传参，都是固定的值。

**混合模式创建（构造 + 原型）**

用构造函数创建对象，会有重复定义方法的问题，用原型创建属性不能传参，所以这里结合一下，构造函数中放属性和一直改变的方法，原型中存储共享的属性和方法。

function Person(name) {

this.name = name;

this.eat = 18;

}

Person.prototype.eat = function() {

console.log('吃饭');

};

// 实例对象

var s1 = new Person('小明');

var s2 = new Person('如花');

console.log(s1.name, s2.name); // 小明 如花

console.log(s1.eat === s2.eat); //true;

**案例**：面向对象的选项卡

原则：先写出普通的写法，然后改成面向对象写法

1、普通方法变型

尽量不要出现函数嵌套函数

可以有全局变量

把onload中不是赋值的语句放到单独函数中（init）

2、改成面向对象（）

先写构造函数

onload中创建对象，并init调用

全局变量就是属性

函数就是方法

(属性和方法前面，都要加this)

改this指向问题（尽量让this指向对象）

注意：this指向是这里的重点

演示时先把所有的this指向改了，然后再看报错，一步一步改。要一步一步分析this都指向谁了。

在事件或者定时器调用时，this指向特别容易出错。一定要注意（复习this的指向）

另外，还要注意event对象，它只能出现在事件函数里面。

阻止默认事件，也要出现在事件函数中。

结论：我们改成面向对象了之后，感觉是不是更复杂了？确实是这样，确实是更复杂了，但是我们这个面向对象特别适合复杂的开发，对于简单的，不太推荐使用面向对象。面对复杂开发时，它特别容易扩展，同时，复用性特别强。上面的例子，多添加几个，就可以发现特别方便复用和扩展。

instanceof

constructor